

① はじめに

前回は1文字の出現頻度を調べましたが、今回はそれを拡張し、複数文字の出現頻度を調べてみたいと思います。

調査対象は、前回と同様 Lewis Carroll の「Alice's Adventures in Wonderland」と Herbert George Wells の「The Time Machine」とします。

- Alice's Adventures in Wonderland ... <http://www.gutenberg.org/etext/11>
- The Time Machine ... <http://www.gutenberg.org/etext/35>

Project Gutenberg で配布されている文書には、ファイルの最初と最後に License 関連の文言等が含まれていますので、あらかじめ当該部分を削除しておきます。

② 文字列の数値化

1文字の場合は、各アルファベットの出現回数をリスト $L[1] \sim L[26]$ に蓄積していきました。各アルファベットのアスキー番号が

a	b	c	d	e	f	g	h	i	j	k	l	m
97	98	99	100	101	102	103	104	105	106	107	108	109
n	o	p	q	r	s	t	u	v	w	x	y	z
110	111	112	113	114	115	116	117	118	119	120	121	122

であることから^{*1}、リスト・インデックスとの対応関係は

$$\text{「リスト・インデックス」} = \text{「アスキー番号」} - \text{「96」}$$

とすればよく、いたって単純でした^{*2}。

長さ n のアルファベット列 (aaa...a ~ zzz...z) の場合は、リスト $L[1] \sim L[26^n]$ に出現回数を蓄積します。そのためには、a を 0、b を 1、...、z を 25 と見なし、26進数の考え方を利用します。すなわち、長さ n のアルファベット列 $a_1a_2 \dots a_n$ の10進数への対応を次式により定義します。

$$\sum_{k=1}^n \{\text{cint}(a_k) - 97\} 26^{n-k} = \{\text{cint}(a_1) - 97\} 26^{n-1} + \{\text{cint}(a_2) - 97\} 26^{n-2} + \dots + \{\text{cint}(a_{n-1}) - 97\} 26^1 + \{\text{cint}(a_n) - 97\} 26^0 \quad [1]$$

^{*1} Maxima では、関数 `cint` によりアルファベットのアスキー番号を出力することが出来ます。

^{*2} そもそも、「96を引く操作」は本質的ではありません。

例えば、長さ 6 のアルファベット列「maxima」なら、

$$(109 - 97) \times 26^5 + (97 - 97) \times 26^4 + (120 - 97) \times 26^3 \\ + (105 - 97) \times 26^2 + (109 - 97) \times 26^1 + (97 - 97) \times 26^0 = 142986480$$

となります。

逆に、10 進数 N をアルファベット列 $a_1 a_2 \dots a_n$ に変換するには、26 で割り算を繰り返していきます：

1. $N \div 26$ を計算し、商 d_1 と余り r_1 を求め、`ascii($r_1 + 97$)` を a_n とおく。
2. $d_1 \div 26$ を計算し、商 d_2 と余り r_2 を求め、`ascii($r_2 + 97$)` を a_{n-1} とおく。
- ...

以下、この操作を商が 0 になるまで繰り返せば「ほぼ」完了です。例えば、10 進数 1372 の場合は、

$$\begin{array}{lll} 1371 \div 26 = 52 \dots 19 & \rightarrow & \text{ascii}(19 + 97) = t \\ 52 \div 26 = 2 \dots 0 & \rightarrow & \text{ascii}(0 + 97) = a \\ 2 \div 26 = 0 \dots 2 & \rightarrow & \text{ascii}(2 + 97) = c \end{array}$$

より、アルファベット列は `cat` となります。ただし、この方法では「a」で始まるアルファベット列を適切に処理することが出来ません³。この問題を回避するため、アルファベット列の長さ（アルファベットの個数）情報を活用します。具体的には、「26 による割り算の繰り返し操作」を実行し、その「繰り返し回数」が「想定されるアルファベット列の長さ」より少なければ、不足分だけ先頭を「a」で埋めます。

```
1 to_alphabet(N, n) := block([q: N, r, L: []], - to_alphabet.mac -
2   while(q # 0) do (
3     [q, r]: divide(q, 26),
4     L: cons(r, L)
5   ),
6   L: append(makelist(0, i, 1, n), L),
7   return(map(ascii, rest(L, length(L)-n) + 97))
8 );
```

Maxima で実装する際は、各割り算ごとに余りをリストに蓄積し（4 行目）、割り算の繰り返し回数の如何に関わらず、0 で埋め尽くしたリストを先頭に添加し（6 行目）、最後に超過し

³ 「a」は 0 に対応しているので、例えば「anode」と「node」を 10 進数に変換すると、いずれも 238034 となり、この 10 進数からものとアルファベット列を特定することは不可能です。

た個数を除去して（必要な個数だけを）出力します（7行目）。この関数の実行例は下記の通りです。

```
(%i1) load("to_alphabet.mac");
(%o1) to_alphabet.mac
(%i2) to_alphabet(238034, 4);
(%o2) [n, o, d, e]
(%i3) to_alphabet(238034, 5);
(%o3) [a, n, o, d, e]
```

③ アルファベット列の出現回数

Maximaを使って、文章ファイル「file」と整数「n」を引数にとり、「file」に含まれる「長さnのアルファベット列」の個数を集計する関数を作成します。

```
1 count_g(file, n) := block([a: [], s: [], m,                                - count_g.mac -
2   L: makelist(0, i, 1, 26^n)],
3   for i in read_list(file) do (
4     a: charlist(sdowncase(sprintf(false, "~a", i))),
5     for j in a do (
6       if alphacharp(j) then (
7         if length(s) = n-1 then (
8           s: endcons(cint(j)-97, s*26),
9           m: apply("+", s),
10          L[m+1]: L[m+1] + 1,
11          s: rest(s, 1)
12        ) else (
13          s: endcons(cint(j)-97, s*26)
14        )
15      )
16    )
17  ),
18  return(L)
19 );
```

2行目の変数Lは、各アルファベット列の個数を集計するためのリストです。

3行目の関数read_listにより、ファイル「file」を1行毎に分解し、更に、4行目の関数charlistにより、各行を1字毎に分解します。

分解された文字には半角スペースやピリオド等のアルファベット以外の文字も含まれるため、それらを除外した上で（6行目のif文）、長さが n になるまでリスト s に蓄積を続け（13行目）、長さが n になったら数え上げ（10行目）、リスト s の先頭から1文字取り除く処理を繰り返します。

リスト s への蓄積処理（8行目と13行目）では、ほんの少しだけ工夫をしています。素朴に、`cint(j)-97`を蓄積していく（`s: endcons(cint(j)-97, s)`）と、数値化処理は $m: \text{sum}(s[k]*26^{(n-k)}, k, 1, n)$ のようになり、毎回 26^{n-k} を計算することになります。これに対し、上記の手順のように26倍ずつずらして蓄積していけば、数値化処理が（少しだけ）速くなることが期待出来ます。

なお、前節では、アルファベット列を26進数と見なして数値化したため、対応する数値は「0以上 26^n 未満」となりますが、Maximaのリスト・インデックスは（普通は）1から始まるため、10行目で「数値化されたアルファベット列に1を加えた数」を「リスト・インデックス」にしています。

この関数を利用して、「Alice's Adventures in Wonderland」と「The Time Machine」について、連続する2文字（長さ2のアルファベット列）の出現回数を調べ、相関係数を計算してみましょう。

```
(%i1) load("count_g.mac")$

(%i2) load("numericalio")$

(%i3) A: count_g("11.txt", 2);
(%o3) [9, 251, 208, 477, 0, 93, 204, 48, 720, 15, 130, 1047, 232, 1622,
..... (後略)

(%i4) T: count_g("35.txt", 2);
(%o4) [9, 326, 648, 655, 5, 142, 263, 33, 474, 2, 119, 901, 481, 2348,
..... (後略)

(%i5) correl(L1, L2) := block([x1, x2],
  x1: apply("+", L1)/length(L1),
  x2: apply("+", L2)/length(L2),
  sum((L1[i] - x1)*(L2[i] - x2), i, 1, length(L1))
    /sqrt(sum((L1[i] - x1)^2, i, 1, length(L1)))
    /sqrt(sum((L2[i] - x2)^2, i, 1, length(L2)))
)$

(%i6) correl(A, T), numer;
(%o6) .9659875128207454
```

約0.97ですから、極めて強い相関が有ると言えるでしょう。なお、上記の出力(%o3)と

(%o4) は、順に「aa」、「ab」、「ac」、... の出現回数です*4。

出現回数トップ 10 は次の通りです。

```
(%i8) top_10(L) := block([L1: [], L2: []],
  L1: makelist([L[i], i-1], i, 1, length(L)),
  L2: sort(L1, ordergreatp),
  for i: 1 thru 10 do
    print([to_alphabet(L2[i][2], 2), L2[i][1]])
)$
```

```
(%i9) top_10(A);
[[h, e], 3789]
[[t, h], 3666]
[[i, n], 2039]
[[e, r], 1993]
[[a, n], 1622]
[[o, u], 1570]
[[i, t], 1360]
[[n, d], 1293]
[[h, a], 1277]
[[t, o], 1256]
(%o9) done
```

```
(%i11) top_10(T);
[[t, h], 4806]
[[h, e], 3873]
[[i, n], 2958]
[[a, n], 2348]
[[e, r], 2301]
[[r, e], 2048]
[[n, d], 2013]
[[e, d], 1845]
[[e, s], 1799]
[[s, t], 1728]
(%o11) done
```

トップ 5 くらいまでがほぼ一致しているようです。

④ 接続文字の出現頻度

前節の結果（長さ 2 のアルファベット列の出現回数）を利用し、接続文字の出現回数を出力してみます。前回の結果から、出現率の一番高いアルファベットは「e」ですから、「e」の次の文字の出現回数を出力してみます。

*4 結果が出力されるまで、かなり時間がかかります。Core 2 Duo 2.66 GHz で 14 秒ほどです。

```
(%i12) next_char(L, c) := block([L1: [], L2: []],
  L1: makelist([L[i], i-1], i, 1, length(L)),
  L2: sort(L1, ordergreatp),
  for i in L2 do
    if quotient(i[2], 26) = cint(c)-97 then
      print([to_alphabet(i[2], 2), i[1]])
  )$

(%i13) next_char(A, "e");
[[e, r], 1993]
[[e, a], 1251]
[[e, d], 1204]
[[e, s], 1180]
[[e, n], 1052]
... (後略)

(%o13)                                     done

(%i14) next_char(T, "e");
[[e, r], 2301]
[[e, d], 1845]
[[e, s], 1799]
[[e, n], 1599]
[[e, a], 1481]
... (後略)

(%o14)                                     done
```

「Alice's Adventures in Wonderland」(リスト A) も「The Time Machine」(リスト T) も、「e」の次の文字は「r」である確率が最も高いという結果でした。

⑤ 念のため補足

関数 `count_g` は、一応任意の長さのアルファベット列の出現回数を集計出来るように定義してありますが、長さはせいぜい3までにしておくことをお勧めします。

また、Maxima に拘らずに通常のプログラミング言語を利用する方がよいと思います。